Contents

Remotely Accessing Telemetry Addons

6 minute read page test

This task shows how to configure Istio to expose and access the telemetry addons outside of a cluster.

Configuring remote access

Remote access to the telemetry addons can be configured in a number of different ways. This task covers two basic access methods: secure (via HTTPS) and insecure (via HTTP). The secure method is *strongly recommended* for any production or sensitive environment. Insecure access is simpler to set up, but will not protect any credentials or data transmitted outside of your cluster.

For both options, first follow these steps:

1. Install Istio in your cluster.

   To additionally install the telemetry addons, follow the integrations documentation.

2. Set up the domain to expose addons. In this example, you expose each addon on a subdomain, such as `grafana.example.com`.

   - If you have an existing domain pointing to the external IP address of `istio-ingressgateway` (say example.com):

   ```
   $ export INGRESS_DOMAIN="example.com"
   ```

   - If you do not have a domain, you may use

`nip.io` which will automatically resolve to the IP address provided. This is not recommended for production usage.

```
$ export INGRESS_HOST=$(kubectl -n istio-system get service
 istio-ingressgateway -o jsonpath='{.status.loadBalancer.in
gress[0].ip}')
$ export INGRESS_DOMAIN=${INGRESS_HOST}.nip.io
```

Option 1: Secure access (HTTPS)

A server certificate is required for secure access.

Follow these steps to install and configure server certificates for a domain that you control.

This option covers securing the transport layer *only*. You should also configure the telemetry addons to require authentication when exposing them externally.

This example uses self-signed certificates, which may not be appropriate for production usages. For these cases, consider using `cert-manager` or other tools to

provision certificates. You may also visit the Securing Gateways with HTTPS task for general information on using HTTPS on the gateway.

1. Set up the certificates. This example uses `openssl` to self sign.

```
$ CERT_DIR=/tmp/certs
$ mkdir -p ${CERT_DIR}
$ openssl req -x509 -sha256 -nodes -days 365 -newkey rsa:20
48 -subj "/O=example Inc./CN=*.${INGRESS_DOMAIN}" -keyout $
{CERT_DIR}/ca.key -out ${CERT_DIR}/ca.crt
$ openssl req -out ${CERT_DIR}/cert.csr -newkey rsa:2048 -n
odes -keyout ${CERT_DIR}/tls.key -subj "/CN=*.${INGRESS_DOM
AIN}/O=example organization"
$ openssl x509 -req -days 365 -CA ${CERT_DIR}/ca.crt -CAkey
 ${CERT_DIR}/ca.key -set_serial 0 -in ${CERT_DIR}/cert.csr
-out ${CERT_DIR}/tls.crt
$ kubectl create -n istio-system secret tls telemetry-gw-ce
rt --key=${CERT_DIR}/tls.key --cert=${CERT_DIR}/tls.crt
```

2. Apply networking configuration for the telemetry addons.

    1. Apply the following configuration to expose

Grafana:

```
$ cat <<EOF | kubectl apply -f -
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: grafana-gateway
  namespace: istio-system
spec:
  selector:
    istio: ingressgateway
  servers:
  - port:
      number: 443
      name: https-grafana
      protocol: HTTPS
    tls:
      mode: SIMPLE
      credentialName: telemetry-gw-cert
    hosts:
```

```
    - "grafana.${INGRESS_DOMAIN}"
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: grafana-vs
  namespace: istio-system
spec:
  hosts:
  - "grafana.${INGRESS_DOMAIN}"
  gateways:
  - grafana-gateway
  http:
  - route:
    - destination:
        host: grafana
        port:
          number: 3000
---
apiVersion: networking.istio.io/v1alpha3
```

```
kind: DestinationRule
metadata:
  name: grafana
  namespace: istio-system
spec:
  host: grafana
  trafficPolicy:
    tls:
      mode: DISABLE
---
EOF
gateway.networking.istio.io/grafana-gateway created
virtualservice.networking.istio.io/grafana-vs created
destinationrule.networking.istio.io/grafana created
```

2. Apply the following configuration to expose Kiali:

```
$ cat <<EOF | kubectl apply -f -
```

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: kiali-gateway
  namespace: istio-system
spec:
  selector:
    istio: ingressgateway
  servers:
  - port:
      number: 443
      name: https-kiali
      protocol: HTTPS
    tls:
      mode: SIMPLE
      credentialName: telemetry-gw-cert
    hosts:
    - "kiali.${INGRESS_DOMAIN}"
---
apiVersion: networking.istio.io/v1alpha3
```

```
kind: VirtualService
metadata:
  name: kiali-vs
  namespace: istio-system
spec:
  hosts:
  - "kiali.${INGRESS_DOMAIN}"
  gateways:
  - kiali-gateway
  http:
  - route:
    - destination:
        host: kiali
        port:
          number: 20001
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: kiali
```

```
  namespace: istio-system
spec:
  host: kiali
  trafficPolicy:
    tls:
      mode: DISABLE
---
EOF
gateway.networking.istio.io/kiali-gateway created
virtualservice.networking.istio.io/kiali-vs created
destinationrule.networking.istio.io/kiali created
```

3. Apply the following configuration to expose Prometheus:

```
$ cat <<EOF | kubectl apply -f -
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
```

```
  name: prometheus-gateway
  namespace: istio-system
spec:
  selector:
    istio: ingressgateway
  servers:
  - port:
      number: 443
      name: https-prom
      protocol: HTTPS
    tls:
      mode: SIMPLE
      credentialName: telemetry-gw-cert
    hosts:
    - "prometheus.${INGRESS_DOMAIN}"
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: prometheus-vs
```

```
  namespace: istio-system
spec:
  hosts:
  - "prometheus.${INGRESS_DOMAIN}"
  gateways:
  - prometheus-gateway
  http:
  - route:
    - destination:
        host: prometheus
        port:
          number: 9090
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: prometheus
  namespace: istio-system
spec:
  host: prometheus
```

```
  trafficPolicy:
    tls:
      mode: DISABLE
---
EOF
gateway.networking.istio.io/prometheus-gateway created
virtualservice.networking.istio.io/prometheus-vs create
d
destinationrule.networking.istio.io/prometheus created
```

4. Apply the following configuration to expose the tracing service:

```
$ cat <<EOF | kubectl apply -f -
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: tracing-gateway
  namespace: istio-system
```

```
spec:
  selector:
    istio: ingressgateway
  servers:
  - port:
      number: 443
      name: https-tracing
      protocol: HTTPS
    tls:
      mode: SIMPLE
      credentialName: telemetry-gw-cert
    hosts:
    - "tracing.${INGRESS_DOMAIN}"
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: tracing-vs
  namespace: istio-system
spec:
```

```
  hosts:
  - "tracing.${INGRESS_DOMAIN}"
  gateways:
  - tracing-gateway
  http:
  - route:
    - destination:
        host: tracing
        port:
          number: 80
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: tracing
  namespace: istio-system
spec:
  host: tracing
  trafficPolicy:
    tls:
```

```
      mode: DISABLE
---
EOF
gateway.networking.istio.io/tracing-gateway created
virtualservice.networking.istio.io/tracing-vs created
destinationrule.networking.istio.io/tracing created
```

3. Visit the telemetry addons via your browser.

If you used self signed certificates, your browser will likely mark them as insecure.

- Kiali: `https://kiali.${INGRESS_DOMAIN}`

- Prometheus:

  `https://prometheus.${INGRESS_DOMAIN}`
- Grafana: `https://grafana.${INGRESS_DOMAIN}`
- Tracing: `https://tracing.${INGRESS_DOMAIN}`

Option 2: Insecure access (HTTP)

1. Apply networking configuration for the telemetry addons.
    1. Apply the following configuration to expose

Grafana:

```
$ cat <<EOF | kubectl apply -f -
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: grafana-gateway
  namespace: istio-system
spec:
  selector:
    istio: ingressgateway
  servers:
  - port:
      number: 80
      name: http-grafana
      protocol: HTTP
    hosts:
    - "grafana.${INGRESS_DOMAIN}"
---
apiVersion: networking.istio.io/v1alpha3
```

```
kind: VirtualService
metadata:
  name: grafana-vs
  namespace: istio-system
spec:
  hosts:
  - "grafana.${INGRESS_DOMAIN}"
  gateways:
  - grafana-gateway
  http:
  - route:
    - destination:
        host: grafana
        port:
          number: 3000
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: grafana
```

```
  namespace: istio-system
spec:
  host: grafana
  trafficPolicy:
    tls:
      mode: DISABLE
---
EOF
gateway.networking.istio.io/grafana-gateway created
virtualservice.networking.istio.io/grafana-vs created
destinationrule.networking.istio.io/grafana created
```

2. Apply the following configuration to expose Kiali:

```
$ cat <<EOF | kubectl apply -f -
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
```

```
  name: kiali-gateway
  namespace: istio-system
spec:
  selector:
    istio: ingressgateway
  servers:
  - port:
      number: 80
      name: http-kiali
      protocol: HTTP
    hosts:
    - "kiali.${INGRESS_DOMAIN}"
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: kiali-vs
  namespace: istio-system
spec:
  hosts:
```

```
  - "kiali.${INGRESS_DOMAIN}"
  gateways:
  - kiali-gateway
  http:
  - route:
    - destination:
        host: kiali
        port:
          number: 20001
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: kiali
  namespace: istio-system
spec:
  host: kiali
  trafficPolicy:
    tls:
      mode: DISABLE
```

```
---
EOF
gateway.networking.istio.io/kiali-gateway created
virtualservice.networking.istio.io/kiali-vs created
destinationrule.networking.istio.io/kiali created
```

3. Apply the following configuration to expose Prometheus:

```
$ cat <<EOF | kubectl apply -f -
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: prometheus-gateway
  namespace: istio-system
spec:
  selector:
    istio: ingressgateway
  servers:
```

```
  - port:
      number: 80
      name: http-prom
      protocol: HTTP
    hosts:
    - "prometheus.${INGRESS_DOMAIN}"
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: prometheus-vs
  namespace: istio-system
spec:
  hosts:
  - "prometheus.${INGRESS_DOMAIN}"
  gateways:
  - prometheus-gateway
  http:
  - route:
    - destination:
```

```
        host: prometheus
        port:
          number: 9090
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: prometheus
  namespace: istio-system
spec:
  host: prometheus
  trafficPolicy:
    tls:
      mode: DISABLE
---
EOF
gateway.networking.istio.io/prometheus-gateway created
virtualservice.networking.istio.io/prometheus-vs create
d
destinationrule.networking.istio.io/prometheus created
```

4. Apply the following configuration to expose the tracing service:

```
$ cat <<EOF | kubectl apply -f -
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: tracing-gateway
  namespace: istio-system
spec:
  selector:
    istio: ingressgateway
  servers:
  - port:
      number: 80
      name: http-tracing
      protocol: HTTP
    hosts:
    - "tracing.${INGRESS_DOMAIN}"
```

```
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: tracing-vs
  namespace: istio-system
spec:
  hosts:
  - "tracing.${INGRESS_DOMAIN}"
  gateways:
  - tracing-gateway
  http:
  - route:
    - destination:
        host: tracing
        port:
          number: 80
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
```

```
metadata:
  name: tracing
  namespace: istio-system
spec:
  host: tracing
  trafficPolicy:
    tls:
      mode: DISABLE
---
EOF
gateway.networking.istio.io/tracing-gateway created
virtualservice.networking.istio.io/tracing-vs created
destinationrule.networking.istio.io/tracing created
```

2. Visit the telemetry addons via your browser.

   - Kiali: `http://kiali.${INGRESS_DOMAIN}`
   - Prometheus:

     `http://prometheus.${INGRESS_DOMAIN}`

- Grafana: `http://grafana.${INGRESS_DOMAIN}`
- Tracing: `http://tracing.${INGRESS_DOMAIN}`

Cleanup

- Remove all related Gateways:

```
$ kubectl -n istio-system delete gateway grafana-gateway ki
ali-gateway prometheus-gateway tracing-gateway
gateway.networking.istio.io "grafana-gateway" deleted
gateway.networking.istio.io "kiali-gateway" deleted
gateway.networking.istio.io "prometheus-gateway" deleted
gateway.networking.istio.io "tracing-gateway" deleted
```

- Remove all related Virtual Services:

```
$ kubectl -n istio-system delete virtualservice grafana-vs
kiali-vs prometheus-vs tracing-vs
virtualservice.networking.istio.io "grafana-vs" deleted
virtualservice.networking.istio.io "kiali-vs" deleted
virtualservice.networking.istio.io "prometheus-vs" deleted
virtualservice.networking.istio.io "tracing-vs" deleted
```

- Remove all related Destination Rules:

```
$ kubectl -n istio-system delete destinationrule grafana ki
ali prometheus tracing
destinationrule.networking.istio.io "grafana" deleted
destinationrule.networking.istio.io "kiali" deleted
destinationrule.networking.istio.io "prometheus" deleted
destinationrule.networking.istio.io "tracing" deleted
```